REMARKS

Claims 1-17 are pending in the action. No amendments have been made to the claims.

The Office objects to the drawings stating that they do not include certain reference signs. Correction to the drawings has been made and replacement sheets are submitted herewith.

The Office rejects claims 9-17 under 103 U.S.C. §102(b) as being anticipated by U.S. Patent No. 6,064,751 to Smithies. The Applicant respectfully disagrees and traverses this rejection.

The Smithies reference is directed to a system for capturing and verifying a handwritten signature. In Smithies, a user's signature is captured electronically by a signature capture module 4 and a set of measurements relating to the signature is stored in a signature envelope. The measurements are later compared to a known set of handwritten measurements to verify the identity of the signatory. The verification of a signature is accomplished via a signature verification module 6. The signature capture module and the signature verification module both utilize a set of Application Program Interfaces ("APIs") to permit the incorporation of the signature capture and verification into many different applications. In Smithies, the signature capture module 4 requires the use of a digitizer. Col. 8, lines 28-30.

The Office states that claim 9 and 13 are anticipated under Smithies. The Applicant respectfully contends that the analysis regarding these claims is flawed in that claim 9 is a method claim and claim 13 is a dependent apparatus claim that depends from claim 1. Claim 1 and 9 do not share identical elements. Thus, the analysis proffered by the Office regarding claim 9 is inapplicable to claim 13 as claim 13 depends from a different independent claim. As claim 13 is never specifically analyzed, the Office has provided no basis for the rejection of claim 13 as being anticipated by Smithies, and thus, the Applicant contends that this claim is allowable.

In further support, in its comments of the rejection of claims 1-8, the Office specifically states that the Smithies reference fails to teach "a fitting algorithm, wherein the fitting algorithm is configured to smooth user indicia input into the input pad." Since claim 13 includes this element, claim 13 cannot be anticipated by the Smithies reference per the Office's own admission. As such, the Applicant contends that claim 13 is allowable. Similarly, claims 12, and

14-17 are also allowable, as by the Office's own admission, the Smithies reference does not teach the "fitting algorithm", and each of these claims depends from claims 1 or 5 which require the "fitting algorithm" element, which is not met by the Smithies reference.

Regarding claim 9, the Office states that the Smithies reference discloses a method for receiving and processing user indicia on a network having a user computer, the user computer having "an input device, a display device and a pointer that defines locations on the display device, wherein the input device includes an entry member and is configured to move the pointer in a continuous path on the display device". The Office cites column 4, lines 30-41 in support of this contention. The Applicant respectfully disagrees.

Column 4, lines 30-41 describes a user signing a document. The reference states "[a]s the user signs the document, (e.g., by moving the pen or stylus across the screen), an image appears that traces the movement of the stylus." The claim language states that "an input device includes an entry member and *is configured to move the pointer in a continuous path on the display device.*" The reference fails to teach or suggest two required elements in the claim; namely, (1) a pointer; and (2) that the input device (a) includes an entry member and (b) is configured to move the pointer in a continuous path on the display device. Indeed, "an image appears that traces the movement of the stylus." Nothing in the Smithies reference teaches or even suggests that the pen/stylus is configured to move the pointer. The pointer and pen/stylus (input device) are two separate elements and must both be found in the cited reference.

The Office further states that the Smithies reference teaches "placing the pointer (i.e., pen-based [sic] hardware) within the data receiving region via the input device, depressing the entry member, moving the pointer within the data receiving region via the input device to create the indicia of authorization within the data receiving region (i.e., moving the pen or stylus across the screen)." The Office supports this assertion by again citing column 4, lines 30-41. The Applicant respectfully disagrees.

Assuming the Office is correct, and the pointer is the pen-based hardware, the analysis offered by the Office is flawed as the Office has either (1) assigned the pen-based hardware to *both* the input device and the pointer; or (2) has failed to identify the input device in the claim.

Since these are two separate and distinct elements in the claim, claim construction requires that the reference teach each of these elements separately to anticipate the claim. Further, the Office has failed to identify the entry member and therefore has not identified the step of "depressing the entry member" in the Smithies reference as required by the claim language.

The Office further states that the Smithies reference teaches "applying a fitting algorithm to the user indicia" and cites column 7, lines 44-60 in support of this contention. The Applicant has reviewed the cited portions of the reference, and respectfully states that it does not understand the Office's assertions. Indeed, per the Office's own admission, the Smithies reference does not teach a fitting algorithm. Thus, the Applicant is uncertain how the Smithies reference could teach "applying a fitting algorithm". Further, the cited portions of the reference simply discuss the capture of the signature by a signature capture module and the verification of a signature by passing "the signature envelope 10 to a signature verification module 6." Col. 7, lines 54-55. The signature verification module simply reviews a template database (signatory database) to match the unknown signature with information regarding previously input signatures. Nothing in this portion of the reference identifies, suggests or otherwise hints or discusses a fitting algorithm.

Finally, the Office states that the Smithies reference "present[s] a user an HTML page". The Applicant disagrees and contends that Smithies reference does not "present[] a user an HTML page containing an applet, wherein the applet configures an input pad having a data receiving region on the display device" as required by the claim language.

The Smithies reference states that the signature capture module 4 displays a "form or window 20". See Smithies, col. 10, lines 9-11. There is no indication that this is an HTML page containing an applet that "configures an input pad having a data receiving region". Indeed, the Smithies disclosure states that the "signature capture module 4 ... utilizes a set of APIs (Application Program Interfaces) to permit the incorporation of signature capture ... into many different applications" (col. 8, lines 14-17), thereby teaching away from an HTML page containing an applet.

Further, the signature capture module 4 "requires the availability of both a graphical display device and a digitizer." Col. 8, lines 28-30. This is contrary to the present invention which does not require the use of a digitizer. Indeed, the use of an HTML page containing the applet allows the signature to be captured without the necessity of any further equipment or downloading of software or reference to other software. The Smithies window presupposes a digitizer system as stated in its disclosure, and thus, cannot be an HTML page containing an applet as required by the claim language. This point has been raised by the Applicant in both the prior response to the Office Action and in the Appeal Brief. The Office, to date, has not yet responded to this argument.

In light of the arguments set forth above, the Applicant contends that the Smithies reference fails to teach independent claim 9, and claims 10-11 depending therefrom. Further, claims 12-17 which depend directly or indirectly from claims 1 or 5, are not anticipated by the Smithies reference, in part, for the reasons set forth above with respect to claim 9 and for the reason that the Office has stated that the Smithies reference fails to meet the "fitting algorithm" which is a element of each of these claims. As such, these claims are allowable, and the Applicant respectfully requests withdrawal of this rejection.

The Office further rejects claims 1-8 under 35 U.S.C. §103(a) as being unpatentable over Smithies in view of U.S. Patent No. 5,680,751 to Moussa. The Applicant respectfully disagrees and traverses the rejection.

With respect to independent claims 1 and 5, the Office states that the Smithies reference discloses a data receiving device for accepting user indicia of authorization on a network having a user computer, wherein the user computer includes "a display device and a pointer that defines locations on the display device (col. 4, lines 30-41), comprising: an input device, wherein the input device is configured to control the pointer in the computer and configured to move the pointer in a continuous path on the display device....(col. 4, lines 15-41)" The Applicant respectfully disagrees.

As discussed above, the Office has failed to identify both the "pointer" and the "input device". Column 4, lines 15-41 describes that the client application "calls the signature capture module, which will display on the screen a signature capture window." Col. 4, lines 16-18. The remaining description discusses a user signing a document as discussed above. For the reasons discussed above, the Smithies reference fails to meet the limitation of an input device.

The Office further states that the data processor comprises a software applet, "wherein the software applet configures an input pad comprising a data receiving region, the data receiving region being defined by a matrix grid". In contrast to the Office's assertion, there is no indication that Smithies teaches or suggests a data processor that comprises a *software applet* that "configures an input pad comprising a data receiving region."

An applet is "a software component that runs in the context of another program, for example, a web browser." See e.g., Wikipedia at en.wikipedia.org/wiki/applet. The applet is self-contained to the extent that it "configures an input pad comprising a data receiving region." In this regard, the applet requires no other software or access to any other software. In vast contrast, the signature capture module 4 and signature verification module 6 "utilize a set of APIs (Application Program Interfaces) to permit the incorporation of signature capture and verification into many different applications." Col. 8, lines 14-17. An application programming interface is defined as "the interface that a computer system, library or application provides in order to allow requests for services to be made of it by other computer programs, and/or to allow data to be exchanged between them." See e.g., Wikipedia at en.wikipedia.org/wiki/Application_programming_interface. An API is used to describe how a computer application may access a set of functions without requiring access to the source code of the function or library. Thus, the signature capture and signature verification modules are not self contained, nor are they capable of self-execution. Rather, the signature capture and signature verification modules utilize an API to access third party or other source code. In vast contrast to the present invention, the applet does not require any further source code or access to any other code to operate or perform the function it is intended to perform. The applet, as claimed in the claim, configures the input pad without linking or calling other programs. This is advantageous as it allows for ease of use by users and ease of distribution as there is no need to couple the use with other software or require the user to download other software. In this regard, these are vastly different features. As such, Smithies fails to teach an applet that "configures an input pad". As stated above, the Applicant has distinguished the applet from the application program interface and the signature capture module and signature verification modules in both the last response to the Office Action and in the Appeal Brief. However, this argument has not yet been addressed by the Office.

As independent claims 1 and 5 are not rendered obvious by the Smithies reference in combination with the Moussa reference, dependent claims 2-4 and 6-8 are not rendered obvious by this combination. As such, the Applicant contends that these claims are allowable.

The Applicant believes that the claims are now in condition for allowance. As such, the Applicant respectfully requests that the Office withdraw the rejections and pass the claims onto allowance.

Respectfully submitted,

Dated: Sept 13, 2006

[illegible]

Anna M. Vradenburgh
Reg. No. 39,868
KOPPEL, PATRICK & HEYBL
555 St. Charles Drive, Suite 107
Thousand Oaks, CA 91360
Telephone: (805) 373-0060
Fax: (805) 373-0051

Application programming interface

From Wikipedia, the free encyclopedia

An **application programming interface** (**API**) is the interface that a computer system, library or application provides in order to allow requests for services to be made of it by other computer programs, and/or to allow data to be exchanged between them.

Contents

Description

One of the primary purposes of an **API** is to describe how computer applications and software developers may access a set of (usually third party) functions (for example, within a library) without requiring access to the source code of the functions or library, or requiring a detailed understanding of the functions' internal workings. The software which provides the functionality described by the API is said to be an *implementation* of the API. The API itself is abstract, as it is an interface. A *reference implementation* of an API is the implementation created by the designer of the API, or one which other implementations of the API are expected to be compared against.

For example, an API might describe how an application may call an icon-drawing function within a graphics library, for displaying icons in the screen. A programmer can write a program which calls the icon-drawing function described in the API. When compiled the compiler will link against the API. When executed, the program will use the implementation of the API (a library) to draw the icon.

Computer programs often use the operating system's API to allocate memory and access files. Many types of systems and applications provide and implement APIs, such as graphics systems, databases, networks, web services, and even some computer games.

In many instances, an API is often a part of a *Software development kit* (SDK). An SDK may include an API as well as other tools and perhaps even some hardware, so the two terms are not strictly interchangeable.

There are various design models for APIs. Interfaces intended for the fastest execution often consist of sets of functions, procedures, variables and data structures. However, other models exist as well, such as the interpreter used to evaluate expressions in ECMAScript/JavaScript or in the abstraction layer, which relieves the programmer from needing to know how the functions of the API relate to the lower levels of abstraction. This makes it possible to redesign or improve the functions within the API without breaking code that relies on it.

Two general lines of policies exist regarding publishing APIs:

1. Some companies guard their APIs zealously. For example, Sony used to make its official PlayStation 2 API available only to licensed PlayStation developers. This is because Sony wanted to restrict how many people could write a PlayStation 2 game, and wanted to profit from them as much as possible. This is typical of companies who do not profit from the sale of API implementations (in this case, Sony broke-even on the sale of PlayStation 2 consoles and even took a loss on marketing, instead making it up through game royalties created by API licensing). However, PlayStation 3 is based entirely on open and publicly

available APIs.
2. Other companies propagate their APIs freely. For example, Microsoft deliberately makes most of its API information public, so that software will be written for the Windows platform. The sale of the third-party software sells copies of Microsoft Windows. This is typical of companies who profit from the sale of API implementations (in this case, Microsoft Windows, which is sold at a gain for Microsoft).

Some APIs, such as the ones standard to an operating system, are implemented as separate code libraries that are distributed with the operating system. Others require software publishers to integrate the API functionality directly into the application. This forms another distinction in the examples above. Microsoft Windows APIs come with the operating system for anyone to use. Software for embedded systems such as video game consoles generally falls into the application-integrated category. While an official PlayStation API document may be interesting to read, it is of little use without its corresponding implementation, in the form of a separate library or software development kit.

An API that does not require royalties for access and usage is called "open." The APIs provided by Free software (such as all software distributed under the GNU General Public License, for example glibc), are open by definition, since anyone can look into the source of the software and figure out the API. Although usually authoritative "reference implementations" exist for an API (such as Microsoft Windows for the Win32 API), there's nothing that prevents the creation of additional implementations. For example, most of the Win32 API can be provided under a UNIX system using software called Wine.

It is generally lawful to analyze API implementations in order to produce a compatible one. This technique is called reverse engineering for the purposes of interoperability. However, the legal situation is often ambiguous, so that care and legal counsel should be taken before the reverse engineering is carried out. For example, while APIs usually do not have an obvious legal status, they might include patents that may not be used until the patent holder gives permission.

See also

- Application binary interface (ABI)
- Ontology (computer science)
- Open Service Interface Definitions (OSID)
- Plugin
- Document Object Model
- Web service

Some example APIs

- The PC BIOS call interface
- Single UNIX Specification (SUS)
- Microsoft Win32 API
- Java Platform, Enterprise Edition APIs
- ASPI for SCSI device interfacing
- Carbon and Cocoa for the Macintosh OS
- DirectX for Microsoft Windows
- Simple DirectMedia Layer (SDL)
- Universal Home API
- LDAP Application Program Interface
- svgalib for Linux and FreeBSD
- Google Maps API
- World of Warcraft API (http://www.wowwiki.com/World_of_Warcraft_API)

External links

- How to design a good API and why it matters-PDF (http://lcsd05.cs.tamu.edu/slides/keynote.pdf)
- gotAPI: Searchable API Aggregation Service (http://www.gotapi.com/)

Retrieved from "http://en.wikipedia.org/wiki/Application_programming_interface"

Categories: Cleanup from September 2006 | Technical communication | Application programming interfaces

Applet

From Wikipedia, the free encyclopedia

An **applet** is a software component that runs in the context of another program, for example a web browser. An applet usually performs a very narrow function that has no independent use. Hence, it is an *app*lication *-let*. The term was introduced in AppleScript in 1993. An applet is distinguished from "subroutine" by several features. First, it executes only on the "client" platform environment of a system, as contrasted from "servlet." As such, an applet provides functionality or performance beyond the default capabilities of its container (the browser). Also, in contrast with a subroutine, certain capabilities are restricted by the container. An applet is written in a language that is different from the scripting or HTML language which invokes it. The applet is written in a compiled language, while the scripting language of the container is an interpreted language, hence, the greater performance or functionality of the applet. Unlike a "subroutine," a complete web component can be implemented as an applet.

Contents

Interfaces

Applets usually have some form of user interface or perform a particular piece of the overall user interface in a web page. This distinguishes them from a program written in a scripting programming language (such as JavaScript) that also runs in the context of a larger, client program, but which would not be considered an applet.

Applets generally have the capability of interacting with and/or influencing their host program, through the restricted security privileges, although they are generally not required to do so.

Attributes

Unlike a program, an applet cannot run independently; an applet usually features display and graphics and often interacts with the human user. However, they are usually stateless and have restricted security privileges. The applet must run in a container, which is provided by a host program, through a plugin, or a variety of other applications including mobile devices that support the applet programming model.

Examples

Common examples of applets are Java applets and Flash movies. Another example is the Windows Media Player applet that is used to display embeded video files in Internet Explorer (and other browsers that support the plugin). Some plugins also allow for displaying various 3D model formats in a web browser, via an applet that allow the view of the model to be rotated and zoomed. Many browser games are applet-based, though some may develop into fully functional applications that require installation.

See also

- Java applet
- Some mathematics applets, at MIT (http://www-math.mit.edu/daimp)

Retrieved from "http://en.wikipedia.org/wiki/Applet"

Categories: Programming paradigms | Technology neologisms